NAG Library Function Document

nag_zero_nonlin_eqns_aa_rcomm (c05mdc)

1 Purpose

nag_zero_nonlin_eqns_aa_rcomm (c05mdc) is a comprehensive reverse communication function that finds a solution of a system of nonlinear equations by fixed-point iteration using Anderson acceleration.

2 Specification

```
#include <nag.h>
#include <nagc05.h>
```

```
void nag_zero_nonlin_eqns_aa_rcomm (Integer *irevcm, Integer n, double x[],
     double fvec[], double atol, double rtol, Integer m, double cndtol,
     Integer astart, Integer iwsav[], double rwsav[], NagError *fail)
```

3 Description

The system of equations is defined as:

$$f_k(x_1, x_2, \ldots, x_n) = 0, \quad k = 1, 2, \ldots, n.$$

This homogeneous system can readily be reformulated as

$$g(x) = x, \quad x \in R^n.$$

A standard fixed-point iteration approach is to start with an approximate solution $\hat{x}_0$ and repeatedly apply the function g until possible convergence; i.e., $\hat{x}_{i+1} = g(\hat{x}_i)$, until $\|\hat{x}_{i+1} - \hat{x}_i\| < \text{tol}$. Anderson acceleration uses up to m previous values of $\hat{x}$ to obtain an improved estimate $\hat{x}_{i+1}$. If a standard fixed-point iteration converges, then Anderson acceleration usually results in convergence in far fewer iterations (and therefore using far fewer function evaluations).

Full details of Anderson acceleration are provided in Anderson (1965). In summary, the previous m iterates are combined to form a succession of least squares problems. These are solved using a QR decomposition, which is updated at each iteration.

You are free to choose any value for m, provided $m \leq n$. A typical choice is $m = 4$.

4 References

Anderson D G (1965) Iterative Procedures for Nonlinear Integral Equations *J. Assoc. Comput. Mach.* **12** 547–560

5 Arguments

Note: this function uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **irevcm**. Between intermediate exits and re-entries, **all arguments other than fvec must remain unchanged**.

1: **irevcm** – Integer * *Input/Output*

On initial entry: must have the value 0.

On intermediate exit: specifies what action you must take before re-entering **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)** with **irevcm unchanged**. The value of **irevcm** should be interpreted as follows:

irevcm $= 1$
Indicates the start of a new iteration. No action is required by you, but **x** and **fvec** are available for printing, and a limit on the number of iterations can be applied.

irevcm $= 2$
Indicates that before re-entry to **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)**, **fvec** must contain the function values $f(\hat{x}_i)$.

On final exit: **irevcm** $= 0$ and the algorithm has terminated.

Constraint: **irevcm** $= 0$, 1 or 2.

Note: any values you return to **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)** as part of the reverse communication procedure should not include floating-point NaN (Not a Number) or infinity values, since these are not handled by **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)**. If your code inadvertently **does** return any NaNs or infinities, **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)** is likely to produce unexpected results.

2: **n** – Integer *Input*

On entry: n, the number of equations.

Constraint: **n** > 0.

3: **x**[**n**] – double *Input/Output*

On initial entry: an initial guess at the solution vector, $\hat{x}_0$.

On intermediate exit: contains the current point.

On final exit: the final estimate of the solution vector.

4: **fvec**[**n**] – double *Input/Output*

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $= 1$, **fvec** must not be changed.

If **irevcm** $= 2$, **fvec** must be set to the values of the functions computed at the current point **x**, $f(\hat{x}_i)$.

On final exit: the function values at the final point, **x**.

5: **atol** – double *Input*

On initial entry: the absolute convergence criterion; see below.

Suggested value: $\sqrt{\epsilon}$, where ϵ is the ***machine precision*** returned by **nag_machine_precision (X02AJC)**.

Constraint: **atol** ≥ 0.0.

6: **rtol** – double *Input*

On initial entry: the relative convergence criterion. At each iteration $\|f(\hat{x}_i)\|$ is computed. The iteration is deemed to have converged if $\|f(\hat{x}_i)\| \leq \max(\mathbf{atol}, \mathbf{rtol} \times \|f(\hat{x}_0)\|)$.

Suggested value: $\sqrt{\epsilon}$, where ϵ is the ***machine precision*** returned by **nag_machine_precision (X02AJC)**.

Constraint: **rtol** ≥ 0.0.

7: **m** – Integer *Input*

On initial entry: m, the number of previous iterates to use in Anderson acceleration. If $m = 0$, Anderson acceleration is not used.

Suggested value: $\mathbf{m} = 4$.

Constraint: $0 \leq \mathbf{m} \leq \mathbf{n}$.

8: **cndtol** – double *Input*

On initial entry: the maximum allowable condition number for the triangular *QR* factor generated during Anderson acceleration. At each iteration, if the condition number exceeds **cndtol**, columns are deleted until it is sufficiently small.

If $\mathbf{cndtol} = 0.0$, no condition number tests are performed.

Suggested value: $\mathbf{cndtol} = 0.0$. If condition number tests are required, a suggested value is $\mathbf{cndtol} = 1.0/\sqrt{\epsilon}$.

Constraint: $\mathbf{cndtol} \geq 0.0$.

9: **astart** – Integer *Input*

On initial entry: the number of iterations by which to delay the start of Anderson acceleration.

Suggested value: $\mathbf{astart} = 0$.

Constraint: $\mathbf{astart} \geq 0$.

10: $\mathbf{iwsav}[\mathbf{14} + \mathbf{m}]$ – Integer *Communication Array*

11: $\mathbf{rwsav}[\mathbf{2} \times \mathbf{m} \times \mathbf{n} + \mathbf{m}^\mathbf{2} + \mathbf{m} + \mathbf{2} \times \mathbf{n} + \mathbf{1} + \mathbf{min}(\mathbf{m}, \mathbf{1}) \times \mathbf{max}(\mathbf{n}, \mathbf{3} \times \mathbf{m})]$ – double *Communication Array*

The arrays **iwsav** and **rwsav** MUST NOT be altered between calls to **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)**.

The size of **rwsav** is bounded above by $3 \times \mathbf{n} \times (\mathbf{m} + 2) + 1$.

12: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE

An error occurred in evaluating the QR decomposition during Anderson acceleration. This may be due to slow convergence of the iteration. Try setting the value of **cndtol**. If condition number tests are already performed, try decreasing **cndtol**.

NE_INT

On entry, **astart** = ⟨*value*⟩.
Constraint: $\mathbf{astart} \geq 0$.

On entry, **n** = $\langle value \rangle$.
Constraint: **n** > 0.

On initial entry, **irevcm** = $\langle value \rangle$.
Constraint: **irevcm** = 0.

On intermediate entry, **irevcm** = $\langle value \rangle$.
Constraint: **irevcm** = 1 or 2.

NE_INT_2

On entry, **m** = $\langle value \rangle$ and **n** = $\langle value \rangle$.
Constraint: $0 \leq$ **m** $\leq$ **n**.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_IMPROVEMENT

The iteration is not making good progress, as measured by the reduction in the norm of $f(x)$ in the last $\langle value \rangle$ iterations.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_REAL

On entry, **atol** = $\langle value \rangle$.
Constraint: **atol** ≥ 0.0.

On entry, **cndtol** = $\langle value \rangle$.
Constraint: **cndtol** ≥ 0.0.

On entry, **rtol** = $\langle value \rangle$.
Constraint: **rtol** ≥ 0.0.

7 Accuracy

There are no theoretical guarantees of global or local convergence for Anderson acceleration. However, extensive numerical tests show that, in practice, Anderson acceleration leads to significant improvements over the underlying fixed-point methods (which may only converge linearly), and in some cases can even alleviate divergence.

At each iteration, **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)** checks whether $\|f(\hat{x}_i)\| \leq \max(\mathbf{atol}, \mathbf{rtol} \times \|f(\hat{x}_0)\|)$. If the inequality is satisfied, then the iteration is deemed to have converged. The validity of the answer may be checked by inspecting the value of **fvec** on final exit from **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)**.

8 Parallelism and Performance

nag_zero_nonlin_eqns_aa_rcomm (c05mdc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

During each iteration, Anderson acceleration updates the factors of a *QR* decomposition and uses the decomposition to solve a linear least squares problem. This involves an additional $O(mn)$ floating-point operations per iteration compared with the unaccelerated fixed-point iteration.

nag_zero_nonlin_eqns_aa_rcomm (c05mdc) does not count the number of iterations. Thus, it is up to you to add a limit on the number of iterations and check if this limit has been exceeded when **nag_zero_nonlin_eqns_aa_rcomm (c05mdc)** is called. This is illustrated in the example program below.

10 Example

This example determines the values $x_1, \ldots, x_4$ which satisfy the equations

$$\begin{aligned} \cos x_3 - x_1 &= 0, \\ \sqrt{1 - x_4^2} - x_2 &= 0, \\ \sin x_1 - x_3 &= 0, \\ x_2^2 - x_4 &= 0. \end{aligned}$$

10.1 Program Text

```
/* nag_zero_nonlin_eqns_aa_rcomm (c05mdc) Example Program.
 *
 * Copyright 2017 Numerical Algorithms Group.
 *
 * Mark 26.1, 2017.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagc05.h>
#include <nagx02.h>

int main(void)
{
  Integer  exit_status = 0, i, n = 4, irevcm = 0, m = 2, astart = 0, icount =
    0, imax = 50, exit_loop = 0;
  double   *fvec = 0, *x = 0, *rwsav = 0;
  Integer  *iwsav = 0;
  double   cndtol = 0.0, rtol, atol;
  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_zero_nonlin_eqns_aa_rcomm (c05mdc) Example Program Results\n");

  if (!(fvec = NAG_ALLOC(n, double)) ||
      !(x = NAG_ALLOC(n, double)) ||
      !(iwsav = NAG_ALLOC(14+m, Integer)) ||
      !(rwsav = NAG_ALLOC(2*m*n+m*m+m+2*n+1+MIN(m, 1)*MAX(n, 3*m), double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* The following starting values provide a rough solution. */
  x[0] = 2.0;
  x[1] = 0.5;
  x[2] = 2.0;
  x[3] = 0.5;
  /* nag_machine_precision (x02ajc).
   * The machine precision
   */
  atol = sqrt(nag_machine_precision);
  rtol = sqrt(nag_machine_precision);

  /* nag_zero_nonlin_eqns_aa_rcomm (c05mdc).
   * Solution of a system of nonlinear equations using Anderson acceleration
   * (reverse communication)
   */

  do
    {
      nag_zero_nonlin_eqns_aa_rcomm(&irevcm, n, x, fvec, atol, rtol, m,
                                    cndtol, astart, iwsav, rwsav, &fail);

      switch (irevcm)
        {
        case 1:
          if (icount == imax)
            {
              printf("Exiting after the maximum number of iterations\n\n");
              exit_loop = 1;
            }
          else
            {
              icount++;
            }
          /* x and fvec are available for printing */
          break;
        case 2:
          fvec[0] = cos(x[2]) - x[0];
          fvec[1] = sqrt(1.0 - x[3]*x[3])- x[1];
          fvec[2] = sin(x[0]) - x[2];
          fvec[3] = x[1]*x[1] - x[3];
          break;
        }
    } while (irevcm != 0 && exit_loop == 0);


  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zero_nonlin_eqns_aa_rcomm (c05mdc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  printf("Final approximate solution after %"NAG_IFMT" iterations\n\n", icount);
  for (i = 0; i < n; i++)
    printf("%12.4f  ", x[i]);

  printf("\n");

  if (fail.code != NE_NOERROR)
    exit_status = 2;

 END:
  NAG_FREE(fvec);
  NAG_FREE(x);
  NAG_FREE(iwsav);
  NAG_FREE(rwsav);
  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_zero_nonlin_eqns_aa_rcomm (c05mdc) Example Program Results
Final approximate solution after 31 iterations

      0.7682        0.7862        0.6948        0.6180
```